Discrete ConvolutionalFixedSum

David Griffin[1 [0000−0002−4077−0005]] and Robert I. Davis[1 [0000−0002−5772−0928]]

Department of Computer Science, University of York, UK
{david.griffin,rob.davis}@york.ac.uk

Abstract. The *ConvolutionalFixedSum* (*CFS*) algorithm provides a generalization of the *UUnifast* and *RandFixedSum* algorithms enabling the generation of vectors of uniformly sampled random values that sum to a specified total, while respecting upper and lower constraints on each individual element. *ConvolutionalFixedSum* provides a foundational technique that is used in the generation of synthetic tasks sets that underpin the performance evaluation of real-time scheduling algorithms and schedulability analysis techniques. The *CFS* algorithm generates continuous values; however, some use cases, for example considering message and packet scheduling on communications networks, require solutions that are constrained to discrete values. Adapting the *CFS* algorithm to the discrete case is non-trivial, with simple rounding to the nearest lattice point producing a non-uniform distribution. In this paper, we present the *Discrete-ConvolutionalFixedSum* (*DCFS*) algorithm, which solves this problem.

Keywords: Task Set Generation · Random Sampling · Real-Time Systems

1 Introduction

ConvolutionalFixedSum (*CFS*) [14] provides a foundational technique that is used in the generation of synthetic tasks sets that underpin the performance evaluation of real-time scheduling algorithms and schedulability analysis techniques. The *CFS* algorithm generates vectors of a specified length, n, comprising uniformly sampled random values that sum to a specified total, t, while respecting a vector of upper constraints, **uc**, and a vector of lower constraints **lc**. Specifically, if the vector returned is **p**, with components p_i for $i = 1 \ldots n$, then $\sum_{i=1}^{n} p_i = t$ and $\forall_i\, lc_i \leq p_i \leq uc_i$. *CFS* ensures that the vectors returned are uniformly distributed over the valid region demarcated by the upper and lower constraints on the individual values and the fixed sum. The *CFS* algorithm was published in 2025, addressing uniformity issues found with an earlier algorithm, *Dirichlet-Rescale* (*DRS*) [12], that aimed to solve the same problem.

In the real-time systems field, the most common use case for the *DRS* and *CFS* algorithms is generating sets of n task utilization values that sum to a fixed total utilization, subject to constraints that arise due to the nature of the problem considered, e.g. mixed-criticality scheduling, multi-core scheduling etc.

These vectors of utilization values are used in constructing the synthetic task sets required for systematic evaluation of scheduling policies, tests, and analyses.

Some of the potential use cases for *CFS* are for discrete sampling problems, rather than continuous ones. For example, work on exact analysis for global scheduling requires the generation of task sets with small integer parameters, enabling the evaluation of analysis techniques with very high complexity [2,6]. This leads to a requirement for discrete solutions. Other examples include the evaluation of scheduling algorithms and schedulability tests for real-time communications networks. Here, the overall network bandwidth considered is allocated to recurring messages that are themselves made up of a number of fixed length packets, again leading to a requirement for discrete solutions. Finally, since *CFS* is a general-purpose algorithm, examples from other domains include the evaluation of different approaches to patient bed allocations on admission to hospital [17], with an integer number of beds available per department.

Given that *CFS* samples from $\mathbb{R}^n$, a simple question to ask is whether this algorithm is appropriate for discrete sampling. This paper shows that naïve approaches to adapting *CFS* to the discrete problem do not necessarily yield a uniform distribution. We therefore present *Discrete-ConvolutionalFixedSum* (*DCFS*), which builds upon *CFS*, enabling the discrete sampling problem to be solved. *DCFS* guarantees that the outputs form a uniform distribution with respect to the set of discrete solutions (lattice points) contained within the valid region demarcated by the constraints and the fixed sum.

1.1 Paper Structure

This paper is organised as follows: Section 2 briefly reviews related work, including other sampling methods and concrete applications where this type of sampling problem is relevant. These include examples from real-time systems as well as other fields. Section 3 introduces a ground truth for the discrete sampling problem in the form of enumeration sampling, which is impractical for larger problems due to its very high complexity. The *DCFS* algorithm is then derived, and the pitfalls of adapting *CFS* to the discrete case documented, showing that a naïve approach may yield a non-uniform distribution. Section 4 provides an evaluation of the methods presented, and a detailed verification, showing that the outputs of *DCFS* form a uniform distribution. Runtime performance characteristics are also examined. Finally, Section 5 provides a summary of the work and concluding remarks.

2 Related Work

In the real-time systems field, Bini and Buttazo [4] identified that existing approaches to evaluating schedulability algorithms for single processor systems were biased due to task utilization values that summed to a fixed value being generated in a non-uniform manner. To address this problem, they proposed the

UUnifast algorithm [5]. The Dirichlet Distribution [21] can also be used to the same effect [12].

Evaluating scheduling algorithms for multiprocessor systems [10] introduces additional requirements, specifically that no individual task can have a utilization greater than 1, even though the total utilization required may be as large as m, where m is the number of processors. Davis and Burns' initial approach to solving this problem, referred to as *UUnifast-Discard* [9], simply discarded values that violated the per task constraint. This is tractable when the number of tasks is relatively high in relation to the total utilization required, otherwise the discard rate becomes prohibitive. Fortunately, in such cases Stafford's *RandFixedSum* algorithm [24], identified by Emberson et al. [11], is applicable.

The advent of mixed-criticality scheduling [7] resulted in additional requirements: the low-criticality utilization of each task cannot exceed its high-criticality utilization. This translates into each variate having a separate upper constraint when generating low-criticality utilization values, assuming that the high-criticality values have been set first. Alternatively, it translates into each variate having a separate lower constraint when generating high-criticality utilization values, assuming that the low-criticality values have been set first. *RandFixedSum* [24,11] is unsuitable for this problem, since it exploits symmetry and therefore mandates that every variate generated must satisfy the same constraint. *UUnifast-Discard* [9] could be adapted to handle individual constraints; however, the resulting discard rate renders this approach intractable for many such problems.

The *Dirichlet-Rescale* (*DRS*) algorithm introduced by Griffin et al. [12] attempted to address the problem of individual constraints, employing repeated rescales to transform a randomly sampled point such that it lies within the region specified by the constraints, while preserving uniformity. However, it was later shown that there were a number of issues with *DRS*, rendering the resulting distribution non-uniform in some cases [14,25].

To address the non-uniformity of *DRS*, Griffin and Davis [14] proposed the *ConvolutionalFixedSum* algorithm. *CFS* works by observing that the region described by a set of n arbitrary constraints can be expressed as the intersection of two $(n-1)$-dimensional simplices lying on a hyperplane in n-dimensional space. With this knowledge, *CFS* employs convolution to calculate the volume of the intersecting region, and then uses the Interpolate-Truncate-Project (ITP) algorithm [20] to construct the Inverse Marginal Cumulative Distribution Function [23] of the uniform distribution over the region. This in turn enables Inverse Transformation Sampling [23] to be used to draw from the uniform distribution. As convolution is an operation that can be approximated, Griffin and Davis [14] provided two implementations of *CFS*: an analytical version with $O(2^n)$ complexity, and a numerical approximation with $O(n^3\, s \log(sn))$ complexity, where n is the number of dimensions, and s is proportional to the accuracy of the approximation.

To verify that the outputs of *CFS* form a uniform distribution, Griffin and Davis [14] used the volume calculations from *CFS* to generate a sensitive statistical test of uniformity, referred to as the *slices* test. The slices test divides the

valid region into a number of slices and compares the point density of these slices. These tests were repeated across multiple test instantiations and metastatistics employed to verify that the empirical distribution across all experiments was as expected. While this approach is robust, one critique is that it used a standard χ^2 test, when it is normally preferable to use the G-test instead [19].

An alternative to the direct sampling approach of *CFS* is the Matrix Hit and Run (MHAR) algorithm [8]. MHAR utilizes Monte-Carlo sampling to generate uniform points on an arbitrary polytope[1]. However, MHAR scales at $O(2^{4n})$ in this use case[2], and due to the use of Monte-Carlo sampling cannot yield truly independent values, which may limit the applicability of the approach.

The problem of sampling with constraints has appeared in many research areas; however, before the advent of the *DRS* and *CFS* algorithms, there was no efficient solution. This resulted in other approaches being used, often without regard for uniformity. While there are very many cases where other approaches were used to provide continuous values, the following are a few examples of where discrete solutions would be useful.

In their work on exact analysis for global scheduling, Baker and Cirinei [2] commented that, *"The jaggedness of the graph is due primarily to the small range of integer values permitted for periods and execution times, which produces only a small number of possible utilization values and makes some values more probable than others."* Later research in this area [6] generated task periods at random, before using an ILP to solve for a set of execution times and hence utilization values with the minimum discrepancy from the desired total. While effective in finding solutions, this approach did not consider the characteristics of the distribution of utilization values so obtained.

In network scheduling, there are scenarios where long messages are split into multiple packets that effectively have a fixed length, either as part of the protocol employed, or for gatewaying onto another network that only supports much shorter messages. For example, transfer of messages from TSN to CAN [26] or the transfer of segmented (multi-packet) messages between CAN networks [1]. In both cases, the evaluation assumed that longer messages were comprised of a fixed number of packets (e.g. 10, 20, 30, 40, ...), without considering the characteristics of the distribution of network utilization values obtained.

Outside of the real-time systems domain, the *DRS* algorithm has been used in the study of algorithms for bed allocation within hospitals [17]. Here, the total number of beds in a hospital was assumed fixed, and the effectiveness of the algorithms proposed evaluated for different allocations of those beds between departments and overflow wards. Discussing their use of the *DRS* algorithm, Li et al. [17] note that, *"rounding is required since the number of inpatient beds is an integer."*

[1] A polytope is a flat sided shape in n-dimensions; a generalization of a polyhedron.

[2] The authors of MHAR state that their algorithm scales at $O(m^4)$, with m being the number of points that define the polytope; the intersection of two n-simplicies has up to 2^n points, hence an overall complexity of $O(2^{4n})$.

The mismatch between *CFS*'s continuous samples and the discrete values required can be bridged by a discretization step, i.e. by rounding the values produced. While this is a straightforward method, as we later show, it does not guarantee a uniform distribution. The extent of the non-uniformity is dependent on the parameters of the problem, and may not necessarily change the conclusions reached in prior research that has taken this approximate approach. Best practice would, however, be to make use of an algorithm that guarantees a uniform distribution, hence making it equally likely that every possible input to the scheduling problem of interest is considered.

2.1 Prior work in Mathematics

Barvinok's algorithm [3] forms the foundational work in this area, providing a means of counting the precise number of lattice points in a polytope. An implementation of Barvinok's Algorithm, called Latte, is openly available[3], along with information about the supporting theory and implementation [18]. Unfortunately, Barvinok's algorithm has complexity that is exponential in n. Initial experiments with the Latte implementation indicated that the runtime of the algorithm grows at approximately $O(3^n)$, which very quickly becomes prohibitive.

Kannan and Vempala [15] provide a theorem about the conditions under which the volume of a polytope approximates well the number of lattice points that it contains. Effectively the polytope must be able to contain a ball of radius $n\sqrt{\log(f)}$ where n is the number of dimensions and f is the number of facets (faces). If this condition holds, then sampling the lattice points within the polytope can be done efficiently by sampling the real points within the polytope and then rounding to the nearest lattice point. Translating to the problem that we are interested in, this condition would require that the extent of the polytope, in terms of a permitted tolerance ϵ on the total t to exceed $n\sqrt{\log(f)}$ times the lattice spacing in each dimension. For $n = 5$ and a maximum lattice spacing of 0.01, the minimum acceptable value of ϵ would be 0.05 or 5%, which is already beyond what may be acceptable for most schedulability analysis experiments. Higher dimensionality and larger lattice spacing, would only exacerbate the issue.

Pak [22] proposed a method of sampling lattice points that chooses a hyperplane that dissects the polytope into two sections, X and Y, as well as Z the intersection between the polytope and the hyperplane. One of X, Y, and Z is then selected using probabilities based on the number of lattice points in each section, computed using Barvinok's algorithm. This process then repeats, either with a reduced number of points in X or Y or reduced dimensionality in Z, until a single point is selected. Overall, this requires a polynomial number of calls to Barvinok's algorithm to evaluate the number of points in each of the smaller and smaller polytopes considered. Unfortunately, due to the complexity and runtime of Barvinok's algorithm, this form of solution is not tractable for the problems that we consider.

[3] `https://www.math.ucdavis.edu/~latte/`

3 Solutions to the Discrete Sampling Problem

In this section we describe potential solutions to the discrete sampling problem. In doing so, we also show why naïve approaches are insufficient.

Definition: The **Discrete Sampling Problem** can be defined as follows: Take as input the number of dimensions n, a total to allocate t, a tolerance ϵ in achieving this total, vectors of upper and lower constraints on each random variate **uc** and **lc**, and a lattice. The lattice is specified via a lattice spacing vector $\mathbf{\Lambda}$, and a base lattice point $\mathbf{\Delta}$ that lies within the valid region. The base lattice point sets the lattice offset with respect to the co-ordinate system. The lattice need not be spaced evenly in each dimension. As output, return a randomly chosen lattice point that is valid, i.e. satisfies the constraints, the elements of which sum to t, within a tolerance of $\pm\epsilon$. Further, every valid lattice point should have an equal probability of being returned.

3.1 Tolerance

In the continuous case, n component values can always be obtained that sum exactly to the required total t, provided only that the problem is a valid one, i.e. with constraints that do not prevent that total from being obtained. In the discrete case; however, a precise total t may not be achievable depending on the specified lattice spacing and offset. For example, with a lattice spacing of $[0.3, 0.4]$, zero offset, lower constraints of zero and no upper constraints, a total of $t = 0.5$ cannot be precisely obtained. A degree of tolerance ϵ is required, permitting totals in the range $[t - \epsilon, t + \epsilon]$. For example, $\epsilon = 0.1$ would permit lattice points where the component values sum to a total in the range $[0.4, 0.6]$, which is possible given the lattice spacing, resulting in two valid lattice points $\langle 0, 0.4\rangle$ and $\langle 0.6, 0\rangle$.

Even if a precise total can be achieved, the number of lattice points that intersect with the hyperplane denoted by that total may be very small. As an example, in a three dimensional system with $t = 1$ and using the lattice spacing $[\frac{1}{5}, \frac{1}{3}, \frac{1}{2}]$, zero offset, lower constraints of zero and no upper constraints, only the lattice points $\langle 1, 0, 0\rangle$, $\langle 0, 1, 0\rangle$ and $\langle 0, 0, 1\rangle$ actually lie on the hyperplane $t = 1$. It is unlikely that this behavior is useful or even what was really intended. In this case, the effect of including a tolerance of $\epsilon = 0.1$, rather than 0, is to increase the number of valid lattice points from 3 to 8, with additional valid lattice points added at $\langle \frac{3}{5}, \frac{1}{3}, 0\rangle$, $\langle \frac{2}{5}, \frac{2}{3}, 0\rangle$, $\langle \frac{2}{5}, 0, \frac{1}{2}\rangle$, $\langle \frac{3}{5}, 0, \frac{1}{2}\rangle$, and $\langle \frac{1}{5}, \frac{1}{3}, \frac{1}{2}\rangle$.

In the typical use cases, generating task utilization values, the tolerance equates to a permitted margin on the total utilization required, for example 0.5 ± 0.01, meaning 50% utilization $\pm 1\%$. For discrete problems, adding a tolerance of at least half of the smallest lattice spacing is typically required to obtain meaningful outputs, without sparse areas appearing in the set of valid lattice points.

3.2 Enumeration Sampling

A basic approach for sampling from a discrete lattice is to simply enumerate all of the lattice points within the valid region, and use this information to derive an appropriate sampling function.

Enumeration sampling can be implemented in a number of ways. By regularising the lattice points (i.e. performing rescaling and translation such that the lattice points lie in $\mathbb{Z}^n$), Barvinok's algorithm [3] can be combined with a numerical inverse method such as the ITP algorithm [20] to perform inverse transform sampling in each dimension. However, Barvinok's algorithm has a high complexity. If we assume that the rescaled bounds of the problem fit within an n-dimensional hyper-rectangle[4] with dimensions denoted by the vector $\mathbf{m}$, then the complexity of Barvinok's algorithm can be expressed as $O(\log(\max(\mathbf{m}))^{n \log n})$ [16]. This means that an approach based on Barvinok's algorithm would have substantially worse complexity than the analytical form of the *CFS* algorithm [14].

Rather than use Barvinok's algorithm, for comparison purposes we implement enumeration sampling via a simple enumeration technique that constructs a list of all valid lattice points. Observing that the vector $\mathbf{m}$ used in the description of the complexity of Barvinok's algorithm denotes the number of lattice points that span the valid region in each dimension, we note that there are $\prod_{i=1}^{n} m_i$ lattice points within the hyper-rectangle that bounds the valid region. As checking if a lattice point $\mathbf{p}$ is within the valid region involves only a trivial comparison against each of the constraints, enumerating every valid lattice point is surprisingly viable, provided that the number of dimensions n is low enough and the lattice spacing is not particularly dense, and hence the values of m_i are small. Once a list of all the valid lattice points has been constructed, sampling from the list can be accomplished in constant time, by uniformly sampling an index for the list and returning the associated lattice point. For problems that require multiple samples from the same set of constraints, the cost of enumerating all valid lattice points can be amortised across all of the samples produced.

The main issue with this simple technique for enumeration sampling is generating the list of all valid lattice points, which has $O(n \prod_{i=1}^{n} m_i)$ time and $O(\prod_{i=1}^{n} m_i)$ memory cost. Hence, when the number of dimensions is high, or the lattice spacing is dense, enumeration sampling becomes intractable. This should not be a surprise, since the problem of counting lattice points within a polytope, such as the valid region in our problem, is known to be NP-hard. (Kannan and Vampala [15] showed that the problem can be reduced to determining if a graph is Hamiltonian, a known NP-hard problem). It would therefore be desirable to be able to draw samples without having to count or enumerate the lattice points.

3.3 Adapting Continuous Sampling

Given that enumeration-based sampling quickly becomes intractable, we aim to adapt a faster continuous sampling method to the discrete sampling problem.

[4] A hyper-rectangle is the generalization of a rectangle to higher dimensions.

The intuition being to generate a point with a fast sampler in the continuous domain and then round the generated point to the nearest lattice point.

In the following, we use the term *bounding box* to refer to a hyper-rectangle centred on a lattice point, such that the bounding box encloses all of the continuous-valued points that have that lattice point as their closest lattice point. In other words, only points whose co-ordinates round to the lattice point are enclosed within the bounding box. Since the lattice has basis vectors that are orthogonal, the closest lattice point to any arbitrary point can be obtained by minimizing the error in each co-ordinate independently, i.e. by rounding to the nearest discrete value given by the lattice spacing and offset. This follows directly from the formula for the Euclidian distance between two arbitrary points $\mathbf{v}$ and $\mathbf{w}$, given by the generalization of the Pythagorean theorem: $\sqrt{(\sum_{i=1,n}(v_i - w_i)^2)}$.

The intuition behind the *DCFS* algorithm is to define an expanded region that contains the bounding boxes of all the valid lattice points, i.e those lattice points that are within the valid region. The *CFS* algorithm is then employed to generate a point uniformly distributed over this expanded region. Since the distribution of points generated by the *CFS* algorithm is uniform over the expanded region, it is also uniform over any subset of that region. Specifically, it is uniform over a subset of the expanded region containing only the bounding boxes of all the valid lattice points. The point generated by the *CFS* algorithm is rounded to the nearest lattice point, and any such lattice points that are outside of the valid region are discarded. In the case of a discarded lattice point, the *CFS* algorithm is called again until a valid lattice point is generated. The valid lattice point is then returned by the *DCFS* algorithm. As a consequence, the *DCFS* algorithm generates lattice points with a uniform distribution over the valid region. In other words, it generates all valid lattice points with the same probability, equal to $1/N$ where N is the number of valid lattice points.

Figure 1 provides an illustrative example of the expansion in 2 dimensions x and y, with a lattice spacing of $[0.1, 0.08]$, target total of $t = 0.8$, and a tolerance of $\epsilon = 0.04$. The lower bounds $(0.1, 0.08)$ correspond to the lattice spacing, and the upper bounds are $(1, 1)$. The solid black lines running diagonally across the figure delimit the extent of the valid range of total values, in other words those lines are $x + y = t + \epsilon = 0.84$ and $x + y = t - \epsilon = 0.76$. The red dashed line represents the target total, $x + y = t = 0.8$. The solid black lines running vertically and horizontally indicate the minimum individual values, $x = 0.1$ and $y = 0.08$ respectively. In other words, the trapezium formed by the solid black lines demarcates the valid region in terms of the values that the total $t = x + y$ could take. Within this valid region, there are 9 lattice points, marked by large green dots at co-ordinates $\langle 0.08, 0.7\rangle$, $\langle 0.16, 0.6\rangle$, $\langle 0.24, 0.6\rangle$,$\langle 0.32, 0.5\rangle$, $\langle 0.4, 0.4\rangle$,$\langle 0.48, 0.3\rangle$, $\langle 0.56, 0.2\rangle$,$\langle 0.62, 0.2\rangle$, and $\langle 0.72, 0.1\rangle$. Each lattice point is at the centre of its bounding box, indicated by black dashed lines. Further, the blue lines demarcate the expanded region that encloses the bounding boxes of all of the valid lattice points. This expanded region increases the tolerance by half of the sum of the lattice spacing, i.e. 0.09, thus the diagonal blue lines are at $x + y = 0.93$ and $x + y = 0.67$. The expanded region also reduces the lower

bounds by half of the lattice spacing, to $x = 0.05$ and $y = 0.04$ respectively, and increases the upper bounds by half of the lattice spacing, to $x = 1.05$ and $y = 1.04$ respectively. (Note, the upper bounds have no effect in this example as they are above the total plus the expanded tolerance of 0.93).

In Figure 1, the 9 valid lattice points are shown color coded according to a heat map representing the number of times that they are generated out of 1,000,000 samples. As expected, there is some small statistical variability (less than 0.5%) around the average frequency at which each lattice point is returned, however, the distribution in this case is uniform.

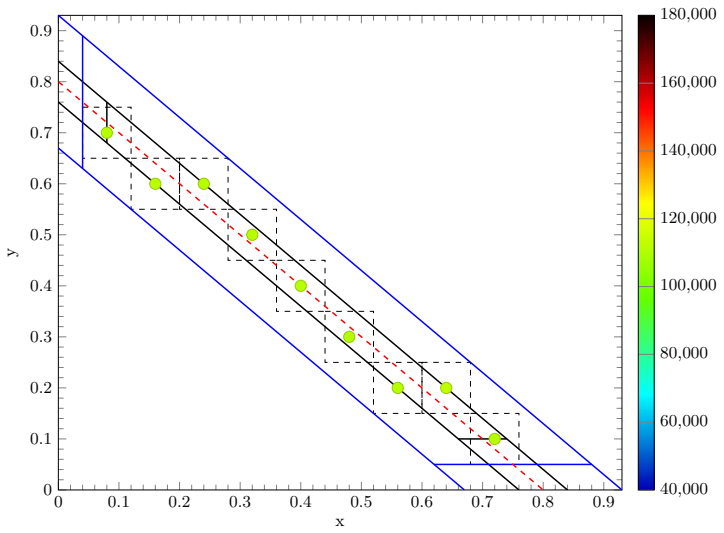


Fig. 1: Illustrative example of expanding the constraints and the tolerance to enclose the bounding boxes of all valid lattice points: Lattice spacing $[0.1, 0.08]$, target total $t = 0.8$ (dashed red line). Tolerance $\epsilon = 0.04$, lower bounds $(0.1, 0.08)$, demarcating the valid region (solid black lines). Tolerance extended by 0.09, lower bounds extended to $(0.05, 0.04)$, demarcating the expanded region (solid blue lines). Valid lattice points (green dots) and their bounding boxes (dashed black lines). Note, the distribution of lattice point frequencies is uniform.

In Figure 1, the bounding boxes of all of the valid lattice points are contained within the expanded region; however, some invalid lattice points (not shown in the figure) can also be generated by rounding, for example, invalid lattice points $\langle 0.24, 0.5\rangle$, $\langle 0.32, 0.4\rangle$, and $\langle 0.4, 0.3\rangle$ among others. This is a side effect of the expansion, and if such a lattice point is generated, then the algorithm should discard it and simply try again. While the bounding boxes of these invalid lattice points do not completely lie within the expanded region, as they are discarded the probability with which they are encountered is irrelevant to the correctness

of the method. There is, however, an obvious performance impact from having to discard some lattice points.

Figure 2 illustrates what happens if a naïve approach is taken and the valid region is not expanded. In this case, the lattice points are not generated following a uniform distribution, but rather their frequency of occurrence reflects the area of the intersection between the relevant bounding box and the valid region. Hence the lattice point $\langle 0.4, 0.4 \rangle$ is heavily over-sampled (136% of the expected frequency), whereas the lattice point $\langle 0.72, 0.1 \rangle$ is heavily under-sampled (43% of the expected frequency).

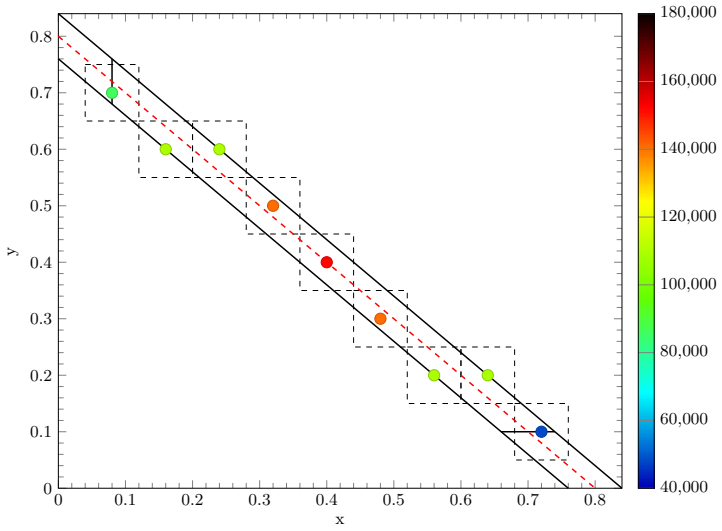


Fig. 2: Naïve use of *CFS* with no expansion of the valid region: Lattice spacing $[0.1, 0.08]$, target total $t = 0.8$ (dashed red line). Tolerance $\epsilon = 0.04$, lower bounds $(0.1, 0.08)$ on the x and y co-ordinates respectively. These constraints demarcate the valid region (solid black lines). Valid lattice points are shown as large color-mapped dots and their bounding boxes (dashed black lines). Note, the distribution of lattice point frequencies is non-uniform.

For efficiency, it is important to limit the amount of expansion as far as possible, since this impacts the algorithm's performance through the generation of invalid lattice points that are then discarded. To achieve the minimum viable expansion, we first round the upper and lower constraints such that the volume of the valid region is minimized, while still containing the same set of valid lattice points. This is achieved by mapping each lower constraint to its ceiling value, and each upper constraint to its floor value in lattice co-ordinates. In each dimension, this aligns the lower constraint with the lowest layer of valid lattice points and the upper constraint with the highest layer. With the valid lattice points now on the edge of the valid region, subsequent expansion by exactly half

of the lattice spacing is the minimum required to ensure that all of the bounding boxes of the valid lattice points are fully contained within the expanded region. (Note, the first step of tightening the constraints is not shown in the example in Figures 1 and 2, which already has the lower constraints aligned to the lattice).

To ensure that it is possible to generate all of the continuously-valued points that round to valid lattice points, it is necessary to expand the tolerance value to counteract the maximum effect of rounding on the continuous values in n dimensions. The worst case occurs when the rounding is of the maximum value possible in each dimension and all of the rounding is in the same direction, either positive or negative. Hence the increase in tolerance required equates to half of the sum of the lattice spacing over all n dimensions.

3.4 *Discrete-ConvolutionalFixedSum* (*DCFS*) Algorithm

To counter the exponential complexity of enumeration-based sampling, the *DCFS* algorithm makes use of *CFS* for continuous sampling, which has the option of a polynomial approximation. Addressing the issues discussed in the previous section, the *DCFS* algorithm is formulated as follows:

1. *DCFS* takes as its input the number of dimensions n, a total to allocate t, a tolerance ϵ on that total, vectors of upper and lower constraints on each random variate $\mathbf{uc}$ and $\mathbf{lc}$, and a lattice. The lattice is specified via a the lattice spacing vector $\mathbf{\Lambda}$ and a base lattice point $\mathbf{\Delta}$ that lies within the valid region. The base lattice point sets the lattice offset with respect to the co-ordinate system.
2. The lower and upper constraints and the tolerance are expanded to encompass a region containing all of the continuously-valued points that round onto any valid lattice point. This is achieved as follows:
   - First, the individual lower and upper constraints are tightened so that, with respect to each dimension, they align with the lowest and highest layer of lattice points demarcated by the original constraints. Then the lower constraints are reduced, and the upper constraints increased by half of the lattice spacing. This ensures that the bounding boxes for each valid lattice point are contained within the expanded constraints, while keeping the expansion to a minimum. Thus: $\mathbf{lc}' \leftarrow \lceil \frac{\mathbf{lc}-\mathbf{\Delta}}{\mathbf{\Lambda}} \rceil \mathbf{\Lambda} + \mathbf{\Delta} - \mathbf{\Lambda}/2$ and $\mathbf{uc}' \leftarrow \lfloor \frac{\mathbf{uc}-\mathbf{\Delta}}{\mathbf{\Lambda}} \rfloor \mathbf{\Lambda} + \mathbf{\Delta} + \mathbf{\Lambda}/2$.
   - Second, the tolerance is increased by half of the sum of the lattice spacing over all n dimensions. This ensures that the expanded region contains all of the continuously-valued points that on subsequent rounding to the nearest valid lattice point reduce or increase their overall sum by up to half of lattice spacing in every dimension, and the revised sum is then within the original tolerance specified. Thus $\epsilon' \leftarrow \epsilon + \frac{\sum \Lambda_i}{2}$.
3. The problem is then cast up one dimension from n to $n+1$, with the extra dimension representing the discrepancy between the sum of the first n components and the required total t. Thus the $(n+1)$-th component of the upper constraint vector, $uc_{n+1} = \epsilon'$, and similarly for the lower constraint vector, $lc_{n+1} = -\epsilon'$

4. *CFS* is then called with the expanded parameters $(n+1,\ t,\ \mathbf{uc}',\ \mathbf{lc}')$ to generate an $(n+1)$-dimensional vector. Ignoring the $(n+1)$-th component, the corresponding n dimensional vector generated in this way is uniformly distributed over the expanded region, with the sum of its component values (i.e. its total) in the range $[t-\epsilon', t+\epsilon']$. (The $n+1$-th component makes up the difference, so that the sum of all $n+1$ components is t).
5. Map the n dimensional vector generated in the previous step to the nearest lattice point by rounding each component value to the nearest discrete value on the lattice.
6. Check if the lattice point is within the valid region originally specified, as demarcated by the original constraints **uc** and **lc** and the permitted range for the total $[t-\epsilon, t+\epsilon]$. If not, then reject the point and goto step 4, calling *CFS* again to generate a new point, otherwise return the valid lattice point.

Algorithm 1 *Discrete-ConvolutionalFixedSum* Algorithm

Input: Number of variates to generate, n, total to allocate, t, lower constraints on variates, **lc**, upper constraints on variates, **uc**, a lattice spacing, $\mathbf{\Lambda}$, a lattice offset, $\mathbf{\Delta}$, a tolerance value, ϵ, and a maximum number of retries, m.

Output: **p**, a uniformly sampled point from the continuous problem described by t, ϵ, **uc**, and **lc**, which lies on the lattice described by $\mathbf{\Lambda}$ and $\mathbf{\Delta}$.

1: $\mathbf{lc}' \leftarrow \lceil \frac{\mathbf{lc}-\mathbf{\Delta}}{\mathbf{\Lambda}} \rceil \mathbf{\Lambda} + \mathbf{\Delta} - \mathbf{\Lambda}/2$ ▷ Expand lower constraints
2: $\mathbf{uc}' \leftarrow \lfloor \frac{\mathbf{uc}-\mathbf{\Delta}}{\mathbf{\Lambda}} \rfloor \mathbf{\Lambda} + \mathbf{\Delta} + \mathbf{\Lambda}/2$ ▷ Expand upper constraints
3: $\epsilon' \leftarrow \epsilon + \frac{\sum \Lambda_i}{2}$ ▷ Expand tolerance
4: append $-\epsilon'$ to $\mathbf{lc}'$ ▷ Append expanded tolerance to lower constraints
5: append ϵ' to $\mathbf{uc}'$ ▷ Append expanded tolerance to upper constraints
6: $c \leftarrow 0$ ▷ Set retries count to 0
7: **while** $c < m$ **do** ▷ Quit if exceed maximum number of retries
8: $\mathbf{p}' \leftarrow CFS(n+1, t, \mathbf{lc}', \mathbf{uc}')$
9: Discard the last element of $\mathbf{p}'$ ▷ Remove value allocated to discrepancy
10: $\mathbf{p} \leftarrow \mathbf{p}'$ round to nearest lattice point described by $\mathbf{\Lambda}$ and $\mathbf{\Delta}$
11: **if** $(\forall i\ lc_i \leq p_i \leq uc_i)$ **and** $abs\left((\sum p_i) - t\right) \leq \epsilon$ **then**
12: **return p**
13: **end if**
14: $c \leftarrow c + 1$
15: **end while**
16: **return** Error - Maximum number of retries exceeded.

We implement *DCFS* via Algorithm 1. Lines 1 and 2 expand the lower and upper constraints on each component. Line 3 computes the expansion needed on the tolerance, represented by dimension $n+1$. This equates to the maximum difference in the sum of the first n dimensions that can be caused by rounding. Lines 4 and 5 set up the lower and upper constraints for the added tolerance dimension, resulting in a new $n+1$ dimensional problem. Line 8 calls *CFS* on the expanded problem in $n+1$ dimensions. Line 9 removes the last element of

the solution returned, since this is the residual value equating to the difference between the sum of the first n elements and the required total t. Line 10 rounds the resulting n-dimensional point to the nearest lattice point, and Line 11 checks if this lattice point is valid. If so, then it is returned, otherwise the point is discarded and the loop (Lines 7 to 15) continues, until either a valid lattice point is found or the maximum number of retries is exceeded. In the latter case, an error message is returned, and the user must decide if they wish to change their problem specification to be more tractable (i.e. with fewer discards), or to allocate more computational resources to sampling and increase the number of retries permitted.

One practical consideration when implementing this algorithm is the use of floating points. Floating points are necessary to express non-integer lattice spacings. However, efficient implementations of the rounding operation on Line 10, are particularly susceptible to floating point inaccuracy. Therefore, care is needed when implementing this step. One effective mitigation is to substitute the first comparison on Line 11 with $\forall_i lc'_i + \frac{\mathbf{\Lambda}}{4} \leq p_i \leq uc'_i - \frac{\mathbf{\Lambda}}{4}$, which still filters lattice points that do not meet the original constraints, while being robust against floating point error.

In general, it is not possible to know exactly what the discard rate will be, without enumerating all of the valid lattice points, which has a very high complexity. It is however possible to derive a very pessimistic bound on the discard rate by observing that for any valid problem, there is at least one valid lattice point. In the worst case, if this is the only such lattice point, then the discard rate is given by the volume of the expanded region divided by the volume defined by the lattice spacing. Considering the analytical complexity, it is possible to construct degenerate problem instances with zero tolerance and only a single solution, i.e. only a single valid lattice point, with an exponentially high discard rate. Such degenerate cases correspond to examples used to prove that the Subset Sum Problem is NP-complete[5]. In practical cases, the discard rate is typically much lower, as illustrated in the evaluation that follows.

4 Evaluation

In this section, we first report on a case study highlighting the pitfalls of converting from continuous to discrete solutions, and the need for both of the expansion steps described in Section 3. We then provide a systematic evaluation of *DCFS* in terms of the uniformity of the results produced, and also examine the runtime performance of the algorithm. The evaluation considers four sampling methods:

Enumeration: Enumerate all valid lattice points, and randomly select one.
Naïve *CFS*: No expansion; generate points using *CFS*, then round to the nearest lattice point.
Intermediate *CFS*: Expand the constraints, but not the tolerance; generate points using *CFS*, then round to the nearest lattice point.
***DCFS*:** Generate lattice points using the *DCFS* algorithm.

[5] `https://en.wikipedia.org/wiki/Subset_sum_problem`

4.1 Case Study

The case study is based on a 3-dimensional problem with lower constraints set to 0, upper constraints set to $(0.9, 0.7, 0.5)$, lattice spacing $[0.1, 0.1, 0.1]$, a required total of $t = 1$, and a tolerance of $\epsilon = 0.1$.

Figure 3 provides a 3-D projection showing the results. The lower constraints, are represented by the blue simplex (triangle), and the upper constraints by the red simplex. The area within these constraints, on the hyperplane $t = x+y+z = 1$ plus or minus the tolerance of 0.1 is the valid region. With the projection used, some of the valid lattice points shown appear to be outside of the valid region. This is however an artefact of the projection used; due to the tolerance value, these lattice points lie on a different hyperplane to the one, with $t = 1$, on which the constraints simplices are drawn.

Figure 3(a) represents, via a heat map, the normalized frequency of each valid lattice point as generated by enumeration sampling. As enumeration sampling simply lists every valid lattice point and then selects one at random for each sample, the resultant distribution is highly uniform, with a normalised frequency of 1. As expected there is still statistical variation in the frequencies; however, this is too small to see with the color map used.

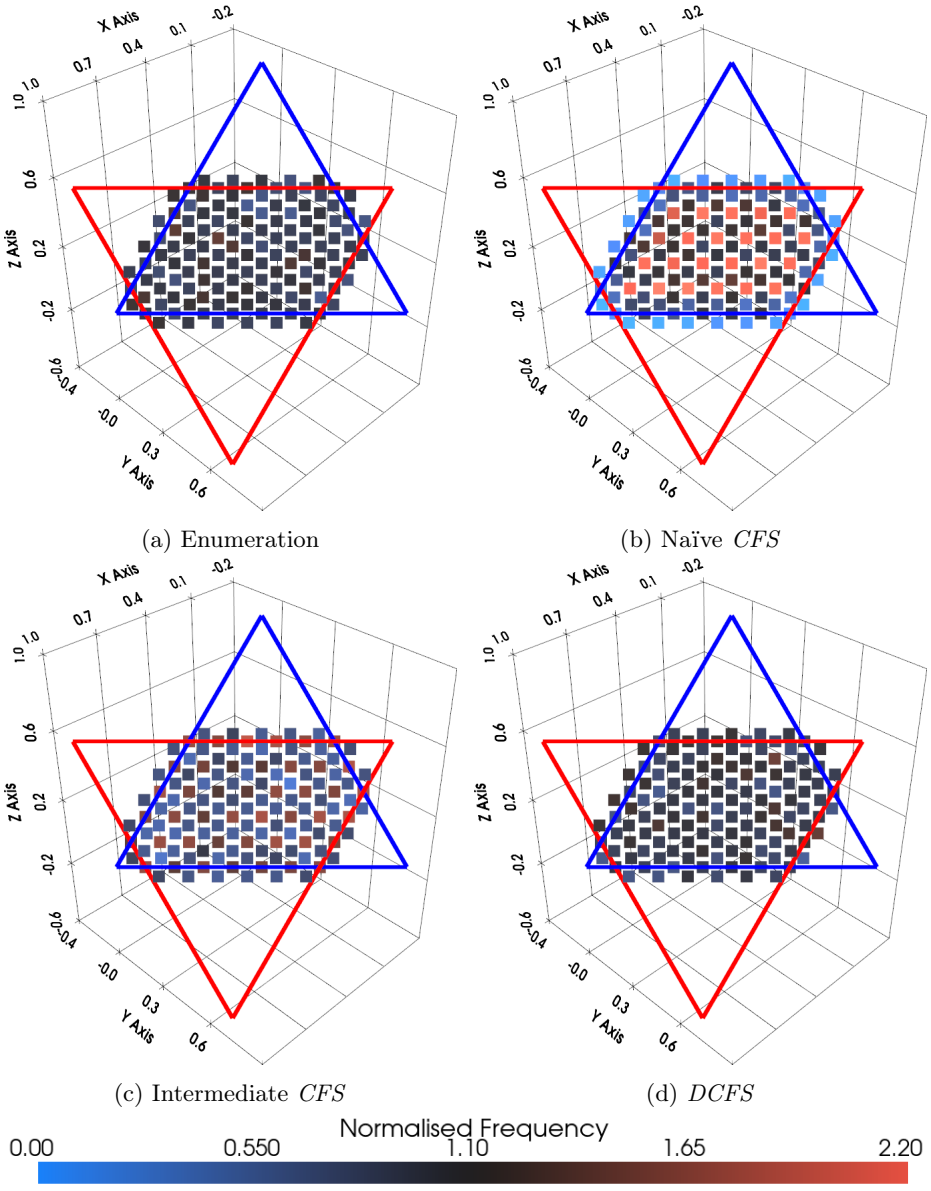
Figure 3(b) shows the results for the naïve *CFS* approach, without any expansion to the region in which continuously-valued points are generated. This clearly illustrates an issue: lattice points near the edge of the region are substantially less likely to be generated. As previously discussed, this is due to these lattice points having bounding boxes only part of which lie within the region that *CFS* generates points on. While lattice points near the edge of the valid region are under-represented, those near the centre of the region are over-represented. The overall distribution is far from uniform.

Figure 3(c) shows the results for the intermediate *CFS* approach with expanded constraints, but no expansion of the tolerance. While the overall distribution is much improved, some lattice points in the centre of the region are oversampled, and many more are slightly undersampled. Careful investigation revealed that the oversampled points lie on the hyperplane $x + y + z = 1$ where the tolerance is 0. Any points lying on the hyperplanes $x + y + z = 0.9$ or $x+y+z = 1.1$, where the tolerance is 0.1, are slightly undersampled, which also explains why not every point in the center of Figure 3(b) is oversampled.

Finally, Figure 3(d) shows the normalised frequency of lattice points sampled via the *DCFS* algorithm. As both of the causes of lattice point bounding boxes exceeding the continuous sampling region are addressed by *DCFS*, we once again have a highly uniform distribution. However, the trade-off compared to naïve *CFS* is an increase in execution time due to the *DCFS* algorithm having to retry when it samples a point outside of the valid region. In this case, approximately 40% of the points generated had to be discarded and a new sample drawn.

4.2 Uniformity Testing

To test uniformity, we built upon the concept of the slices test [14,13] for the continuous sampling problem. The slices test examines the average density of



(a) Enumeration

(b) Naïve *CFS*

(c) Intermediate *CFS*

(d) *DCFS*

Fig. 3: Case study showing the uniform distribution of lattice points produced by sampling via (a) Enumeration and (d) *DCFS*; and the non-uniform distribution of lattice points produced by sampling via (b) Naïve *CFS*, with no expansion and (c) intermediate *CFS* with constraint expansion only.

points within 10 slices of the valid region each with equal volume, with the slicing aligned to the chosen dimension, and the test repeated for all n dimensions. In the continuous case, the equal volumes of the slices ensure that the expected number of points generated in each slice is equal, assuming a uniform distribution. In the discrete case, however, there is the potentially for a highly variable number of valid lattice points within each equal volume slice, which makes the comparison much more complex. To counter this, rather than use equal volume slices, we partitioned the lattice points returned by exhaustive enumeration into 10 groups containing, as far as possible the same number of lattice points[6].

The groupings were aligned to each dimension in turn, using a rotated vector ordering of the lattice points, starting with the component for the chosen dimension. As a simple example of the ordering used, consider the lattice points $\langle 1,2,3\rangle$, $\langle 1,2,4\rangle$, $\langle 1,1,4\rangle$, $\langle 2,2,2\rangle$, $\langle 2,3,2\rangle$. For the first dimension, ordering by the first component, then breaking ties according to the second then third components, results in the ordering: $\langle 1,1,4\rangle$, $\langle 1,2,3\rangle$, $\langle 1,2,4\rangle$, $\langle 2,2,2\rangle$, $\langle 2,3,2\rangle$. For the second dimension, ordering by the second component, then breaking ties according to the third then first components, results in the ordering: $\langle 1,1,4\rangle$, $\langle 2,2,2\rangle$, $\langle 1,2,3\rangle$, $\langle 1,2,4\rangle$, $\langle 2,3,2\rangle$. Finally, for the third dimension, ordering by the third component, then breaking ties according to the first then second components, results in the ordering: $\langle 2,2,2\rangle$, $\langle 2,3,2\rangle$, $\langle 1,2,3\rangle$, $\langle 1,1,4\rangle$, $\langle 1,2,4\rangle$. Once the list of lattice points was ordered, then it was simply partitioned into 10 approximately equal sets. The statistical tests then examined the frequency at which the 10 sets of lattice points were sampled. This *discrete slices* test is analogous to the slices test for the continuous sampling problem, and is similarly sensitive to any non-uniformity around the edges of the valid region in any dimension.

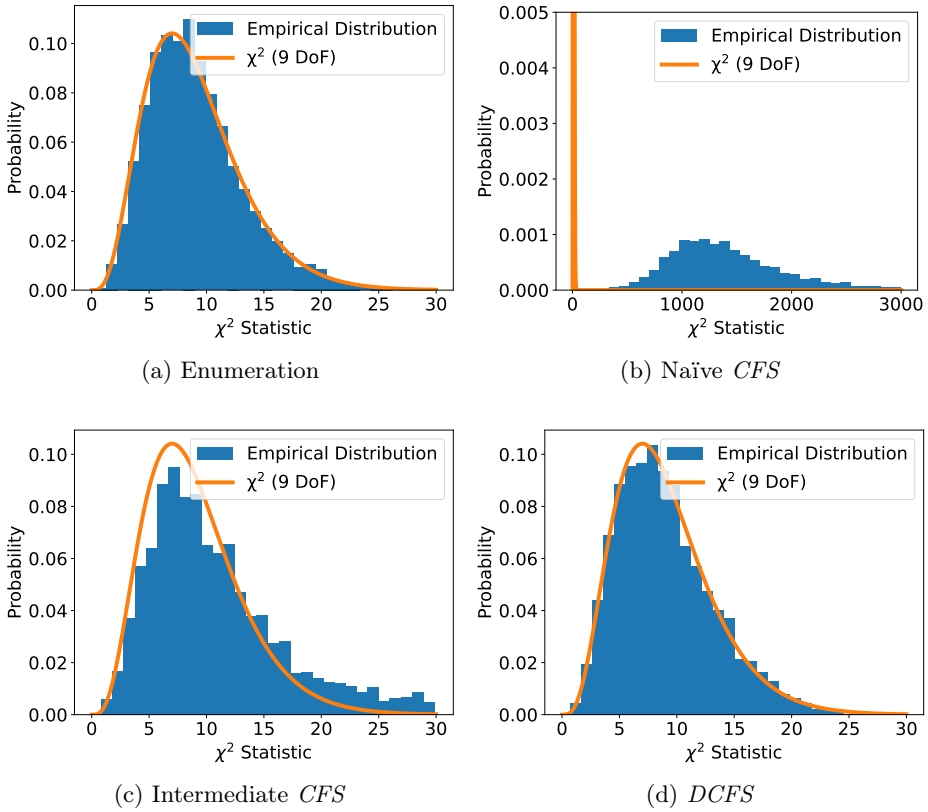
Further, we added an extra ordering based on the total for each lattice point, since due to the tolerance this total can vary. In this case the total was used for initial sorting, with each of the n-dimensions in turn used to break ties. In the running example, this results in the following ordering: $\langle 1,1,4\rangle$, $\langle 1,2,3\rangle$, $\langle 2,2,2\rangle$, $\langle 1,2,4\rangle$, $\langle 2,3,2\rangle$. With the addition of tolerance-based ordering, the discrete slices test is also sensitive to non-uniformity across lattice points with different total values, i.e. residing on different hyperplanes.

Verifying uniformity requires a large number of systematic test cases. To produce these test cases, we generated randomized problems as described below. Without loss of generality, we assumed that all the lower constraints were zero[7]. The number of dimensions, n, was varied between 3 and 10, we generated upper constraints, **uc**, using *UUnifast-Discard*, with these constraints summing to 1.5. For the lattice spacing Λ, we generated values according to the formula $0.2 + 0.3 \cdot uc_i \cdot random()$, where $random()$ returns a value in the range $[0,1]$. This

[6] In the case that the total number of valid lattice points did not exactly divide by 10, then some of the groups had one more lattice point than others.

[7] The implementation of *DCFS* transforms each problem into one where the lower constraints are zero, and reverses this transformation prior to returning the results. Hence starting with the lower constraints set to zero has no impact on uniformity testing, provided that the other parameters are varied appropriately.

corresponds to the hyper-rectangle containing the valid region having between 2 and 5 lattice points along each dimension. For 10 dimensions, this results in at most 5^{10} lattice points, which is approximately 10,000,000. The lattice offset $\mathbf{\Delta}$ was set to $\Lambda \cdot random()$, and finally, the tolerance was randomly chosen between 1 and 3 times the minimum lattice spacing over all of the dimensions.



(a) Enumeration

(b) Naïve *CFS*

(c) Intermediate *CFS*

(d) *DCFS*

Fig. 4: Meta-statistics: discrete slices test results verifying uniformity for sampling via: (a) Enumeration and (d) *DCFS*; and highlighting non-uniformity for sampling via: (b) Naïve *CFS* with no expansion and (c) intermediate *CFS* with constraint expansion only.

For each dimensionality n from 3 to 10, we performed 1,000 randomly generated experiments, with the parameters as described above. For each experiment, we generated 10,000 lattice points, and so each discrete slices test compared against an expected distribution with approximately 1,000 samples per slice. In each experiment, the discrete slices test was applied aligned to each of the n dimensions in turn, and also aligned to the $(n+1)$-th dimension giving the tolerance. With the slices determined and the empirical frequencies recorded, a

χ^2 9 Degrees of Freedom (9-DoF) test was used to determine if the distribution obtained matched that expected. The expected frequency for each discrete slice was given by $S \cdot K_i/N$, where S is the number of samples, N is the total number of valid lattice points, and K_i is the number of lattice points in that slice.

The systematic tests described above resulted in a total of 60,000 χ^2 tests for each of the four methods considered. We employed the meta-statistics approach proposed by Griffin and Davis [14] on this data in order to verify uniformity. The 60,000 χ^2 statistics for each method formed an empirical distribution, which was compared to the theoretical distribution of the 9-DoF χ^2 statistic using a KS-test. Figure 4 shows the empirical distribution for the χ^2 statistic obtained via each of the four methods as a bar chart, overlaid with a line representing the theoretical distribution that it should match.

Figure 4(a) shows the results from using enumeration sampling, and as expected, the empirical data matches the theoretical χ^2 distribution, with a KS-test p-value of 0.08, which is above the 0.05 significance threshold. However, the same is not true of either Figures 4(b) or 4(c), corresponding the Naïve *CFS* and Intermediate *CFS* methods, both of which fail the KS-test with a p-value of 0. These failures are due to substantial and consistent undersampling of edges with Naïve *CFS*, and large excursions from expected sampling frequencies with Intermediate *CFS*. Finally, the results for *DCFS* are shown in Figure 4(d). *DCFS* exhibits the same uniformity properties as enumeration sampling, passing the KS-test with a p-value of 0.38 in this case.

4.3 Performance Testing

The amount of computation required to sample a point with *DCFS* is proportional to two things: the complexity of the associated *CFS* problem, and the number of discards that occur. The complexity of *CFS* is documented [14] as being $O(2^n)$ for the analytical method and $O(n^3\, s \log(sn))$ for the numerical method, where n is the number of dimensions, and s is proportional to the accuracy of the approximation used. Note, using *DCFS*, n is increased by 1 due to the addition of the tolerance. In this subsection, we first examine the discard rate, and then the overall runtime performance of *DCFS*.

For any specific problem, the discard rate is given by the volume of the expanded region divided by the volume of the bounding boxes of all of the valid lattice points, hence, there are three main factors that affect the discard rate:

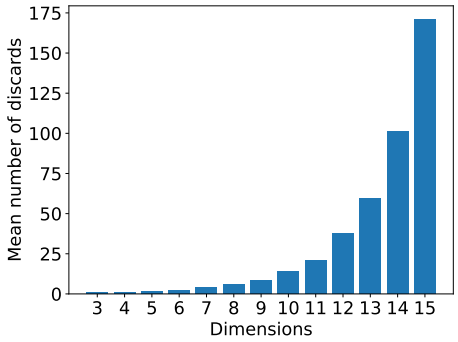
1. The dimensionality n. Expansion of the constraints by half of the lattice spacing in each dimension and expansion of the tolerance by half of the sum of the lattice spacing over all dimensions increases the ratio between these two volumes with increasing n.
2. The lattice spacing in relation to the volume of the valid region. If the lattice spacing is dense, then the expansion of the valid region will be proportionately smaller in relation to its volume.
3. How the lattice spacing divides into the total plus or minus the tolerance, $t \pm \epsilon$. If the lattice spacing divides these values exactly, then this results in

a layer of lattice points along the two hyperplanes given by $t - \epsilon$ and $t + \epsilon$. As a consequence, more of the expanded region is covered by the bounding boxes of the valid lattice points, and so the discard rate is lower. (See Figure 1 in Section 3 for an illustration of how lattice points on these boundaries have more of their bounding boxes extending into the expanded region).

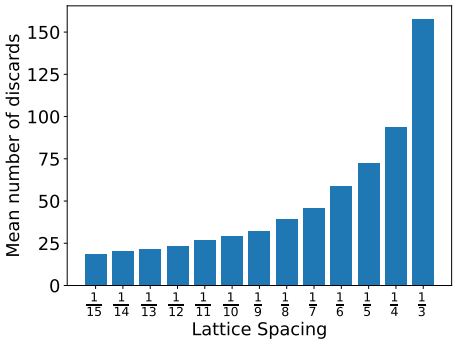
The discard rate can be approximated by the ratio of the volume of the expanded region to that of the valid region, with the latter used to approximate the volume of the bounding boxes of all the valid lattice points. Consider a scenario with a *regular* lattice with equal spacing in each of the n dimensions, with each lower and upper constraints separated by $m\mathbf{\Lambda}$ where $\mathbf{\Lambda}$ is the lattice spacing, and a tolerance $\epsilon = \mathbf{\Lambda}$. The volume of the valid region is proportional to $2m^n$, while the volume of the expanded region is proportional to $(2+n)(m+1)^n$. Hence the discard rate is $O((1+\frac{n}{2})(1+\frac{1}{m})^n)$, i.e. exponential in n, with a growth rate that depends on $(1 + \frac{1}{m})$, where the metric m reflects the lattice density.

To investigate the effect that the three factors listed above have on the discard rate, we carried out experiments conducting a parameter sweep over the number of dimensions and the lattice spacing. In each of the experiments, we set the lower constraints to 0, and specified no upper constraints (i.e. the upper constraints were equal to the total plus the tolerance). Figure 5(a) illustrates how the discard rate varies with n, for a required total $t = 1.0$, and tolerance $\epsilon = 0.1$. The results show the exponential growth in the discard rate as n increases. Figure 5(b) illustrates the discard rate for $n = 10$, a required total $t = 1.0$, and a tolerance $\epsilon = 0$, with the lattice spacing $\mathbf{\Lambda}$ varying from $\frac{1}{15}$ to $\frac{1}{3}$ and hence the lattice density metric m varying from 15 to 3. In this experiment, the lattice spacing always exactly divides the total plus tolerance. The results show how the discard rate increases as the lattice density decreases. Figure 5(c) illustrates the discard rate for $n = 10$, a required total $t = 1.0$, and a tolerance $\epsilon = \frac{1}{9}$, with the lattice spacing $\mathbf{\Lambda}$ varying from $\frac{1}{11}$ to $\frac{1}{9}$ and hence the lattice density parameter m varying from 11 to 9. In this experiment, only the values of $\frac{1}{11}$, $\frac{1}{10}$. $\frac{1}{9}$ for the lattice spacing exactly divide the total. The resulting discard rates for these values (shown in blue) are smaller than those (shown in orange) for the intermediate lattice spacings that do not exactly divide the total. These results show how having layers of lattice points in the expanded region that are close to the two hyperplanes given by $t - \epsilon$ and $t + \epsilon$ result in more of the expanded region being covered by the bounding boxes of valid lattice points and hence a lower discard rate. While the discard rate of *DCFS* increases at small lattice densities, such problems have a relatively small number of valid lattice points and so become more amenable to the enumeration technique. The maximum discards parameter of *DCFS* can be used to check if DCFS is unsuited to a particular problem, and if necessary attempt to use the enumeration technique.

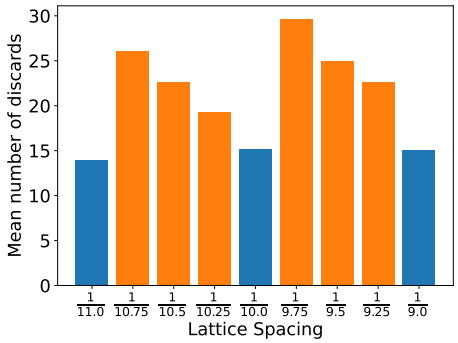
The runtime performance of *DCFS* depends on the performance of the underlying *CFS* algorithm used, either analytical or numerical, and the rate of discards (retries). Figure 5(d) shows the execution time of enumeration sampling and *DCFS*, using the analytical and numerical implementations (signal size $s = 10{,}000$) of *CFS*, using the randomized experiments conducted for uni-



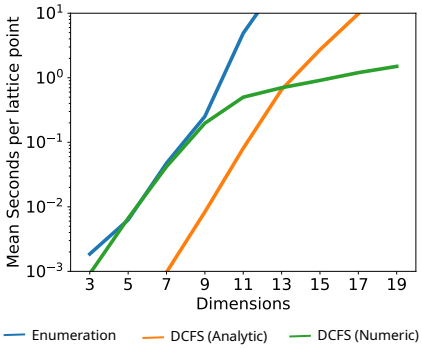
(a) Varying the number of dimensions n, required total $t = 1.0$, lattice spacing $\Lambda = 0.1$ and tolerance $\epsilon = 0.1$, discard rate averaged over 1,000 generated lattice points



(b) Varying the lattice spacing for $n = 10$ dimensions, required total $t = 1.0$, tolerance $\epsilon = 0$, lattice spacing exactly divides the total plus and minus the tolerance



(c) Varying the lattice spacing for $n = 10$ dimensions, required total $t = 1.0$, tolerance $\Lambda = \frac{1}{9}$, showing intermediate points between exactly dividing lattice spacings



(d) Mean runtime required to generate 1 lattice point with randomised constraints and spacing, using the Enumeration method and *DCFS*

Fig. 5: Performance of *DCFS* characterized by the discard rate and runtime.

formity testing. As these experiments use relatively sparse lattices, they are close to the worst case for *DCFS*. This experiment was conducted on a PC with an AMD 9800X3D processor. Here, *DCFS* behaves as expected given the complexity of *CFS* and the number of retries required. Numeric *DCFS* is faster for large n, while analytical *DCFS* is more performant for small n. Enumeration sampling may also be a valid approach for sparse lattices. Enumeration sampling has one advantage over *DCFS* as the complexity is in the enumeration, not the sampling, meaning that sampling any number of lattice points for the exact same problem has virtually the same cost as generating a single lattice point. However, enumeration has the disadvantage of substantially higher base complexity, and a dependency on cache and memory speed due to the amount of data created.

5 Conclusions

Recent work on *ConvolutionalFixedSum* [14] provided a foundational technique that can be used in the generation of synthetic tasks sets that underpin the performance evaluation of real-time scheduling algorithms and schedulability analysis techniques. *CFS* effectively replaced the previous *DRS* algorithm [12] that was aimed at solving the same problem, but later shown to have issues, in that the outputs from *DRS* do not necessarily form a uniform distribution [14,25].

The *CFS* algorithm solves the problem of generating vectors of n component values that sum to a fixed total, subject to individual upper and lower constraints on those values. Further, the vectors produced are uniformly distributed over the valid region demarcated by the constraints and the fixed sum. While *CFS* generates continuous values, there are use cases that require solutions that are constrained to discrete values, i.e. lattice points.

In this paper, we showed that adapting the *CFS* algorithm to the discrete case is non-trivial, with simple rounding to the nearest lattice point producing a non-uniform distribution. To solve this problem, we developed the *Discrete-ConvolutionalFixedSum* (*DCFS*) algorithm. *DCFS* takes the same parameters as *CFS*, along with a lattice spacing and a lattice offset. To address the issue of lattice spacings that do not exactly sum to the target total, *DCFS* also takes a tolerance, allowing vectors to be generated with components that sum to the required total within plus or minus the permitted tolerance.

To test that the outputs from *DCFS* form a uniform distribution over all of the valid lattice points, we performed randomized testing that exhaustively enumerated all of the valid lattice points for each problem and then partitioned them into equal sized groups along each dimension using rotated vector ordering. This formed the *discrete slices* test, analogous to the slices test used to verify *CFS* [14]. We recorded the outputs of *DCFS* against the groups of lattice points, and then used χ^2 tests on each data set, along with meta-statistics on the distribution of the χ^2 statistic to verify uniformity. Finally, we examined how the performance of the *DCFS* algorithm is dependent on the efficiency of the underlying *CFS* algorithm and the rate at which discards occur and hence retries are required.

The *DCFS* algorithm provides complementary technology to the *CFS* algorithm, solving a similar problem, but one where discrete rather than continuous solutions are required. Similar to the *CFS* algorithm, *DCFS* scales sufficiently well to provide a technique that can be used in generating the sets of synthetic tasks or messages required for the performance evaluation of scheduling algorithms and analyses. In this case, where the parameters are such that discrete solutions are required. Given the general purpose nature of the *DCFS* algorithm, it is likely that it will also prove useful in other domains, outside of the real-time systems field.

The source code for the algorithms and tests presented in this paper is available as part of the *ConvolutionalFixedSum* package, available on PyPI and Github [13].

Acknowledgements

This research was funded in part by the MARCH Project (EP/V006029/1), Innovate UK SCHEME project (10065634) and the CHEDDAR Communications hub (EP/Y037421/1, EP/Y036514/1, EP/X040518/1). EPSRC Research Data Management: No new primary data was created during this study.

References

1. Ekain Azketa, J. Javier Gutiérrez, J. Carlos Palencia, Michael González Harbour, Luís Almeida, and Marga Marcos. Schedulability analysis of multi-packet messages in segmented CAN. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation, ETFA 2012, Krakow, Poland, September 17-21, 2012*, pages 1–8. IEEE, 2012. doi:10.1109/ETFA.2012.6489578.
2. Theodore P. Baker and Michele Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal, editors, *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*, volume 4878 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2007. doi:10.1007/978-3-540-77096-1_5.
3. Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4):769–779, 1994. doi:10.1287/moor.19.4.769.
4. Enrico Bini and Giorgio C. Buttazzo. Biasing effects in schedulability measures. In *16th Euromicro Conference on Real-Time Systems (ECRTS 2004), 30 June - 2 July 1004, Catania, Italy, Proceedings*, pages 196–203. IEEE Computer Society, 2004. URL: https://doi.ieeecomputersociety.org/10.1109/ECRTS.2004.7, doi:10.1109/ECRTS.2004.7.
5. Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real Time Syst.*, 30(1-2):129–154, 2005. URL: https://doi.org/10.1007/s11241-005-0507-9, doi:10.1007/S11241-005-0507-9.
6. Artem Burmyakov, Enrico Bini, and Chang-Gun Lee. Towards a tractable exact test for global multiprocessor fixed priority scheduling. *IEEE Trans. Computers*, 71(11):2955–2967, 2022. doi:10.1109/TC.2022.3142540.
7. Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 2018. doi:10.1145/3131347.
8. Mario Vazquez Corte and Luis V. Montiel. Novel matrix hit and run for sampling polytopes and its GPU implementation. *Comput. Stat.*, 40(6):3067–3104, 2025. URL: https://doi.org/10.1007/s00180-023-01411-y, doi:10.1007/S00180-023-01411-Y.
9. Robert I. Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In Theodore P. Baker, editor, *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 398–409. IEEE Computer Society, 2009. doi:10.1109/RTSS.2009.31.
10. Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2011. doi:10.1145/1978802.1978814.

11. Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, July 2010. URL: https://retis.sssup.it/waters2010/waters2010.pdf#page=6.
12. David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 76–88. IEEE, 2020. doi:10.1109/RTSS49844.2020.00018.
13. David Griffin and Robert I. Davis. ConvolutionalFixedSum Software, March 2025. URL: https://github.com/dgdguk/convolutionalfixedsum/, doi:10.5281/zenodo.15107012.
14. David Griffin and Robert I. Davis. Convolutionalfixedsum: Uniformly generating random values with a fixed sum subject to arbitrary constraints. In *31st IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2025, Irvine, CA, USA, May 6-9, 2025*, pages 270–282. IEEE, 2025. doi:10.1109/RTAS65571.2025.00034.
15. Ravi Kannan and Santosh S. Vempala. Sampling lattice points. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 696–700. ACM, 1997. doi:10.1145/258533.258665.
16. Matthias Köppe, Sven Verdoolaege, and Kevin M. Woods. An implementation of the barvinok–woods integer projection algorithm. In Matthias Dehmer, Michael Drmota, and Frank Emmert-Streib, editors, *Proceedings of the 2008 International Conference on Information Theory and Statistical Learning, ITSL 2008, Las Vegas, Nevada, USA, July 14-17, 2008*, pages 53–59. CSREA Press, 2008.
17. Jie Li, Sichen Li, Jun Luo, and Haihui Shen. Simulation optimization for inpatient bed allocation with sharing. *Journal of Systems Science and Systems Engineering*, 2024. doi:10.1007/s11518-024-5625-9.
18. Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *J. Symb. Comput.*, 38(4):1273–1302, 2004. doi:10.1016/j.jsc.2003.04.003.
19. John H McDonald. *Handbook of biological statistics*. Sparky House Publishing, Baltimore, Maryland. USA, 2014. URL: https://www.biostathandbook.com/gtestgof.html.
20. Ivo F. D. Oliveira and Ricardo H. C. Takahashi. An enhancement of the bisection method average performance preserving minmax optimality. *ACM Trans. Math. Softw.*, 47(1):5:1–5:24, 2021. doi:10.1145/3423597.
21. Ingram Olkin and Herman Rubin. Multivariate beta distributions and independence properties of the wishart distribution. *Annals of Mathematical Statistics*, 35(1):261–269, March 1964. doi:10.1214/aoms/1177703748.
22. Igor Pak. On sampling integer points in polyhedra. *Foundations of Computational Mathematics*, pages 319–324, 2002. URL: https://www.math.ucla.edu/~pak/papers/barv2.pdf, doi:10.1142/9789812778031_0013.
23. Murray R Spiegel and Larry J Stephens. *Schaum's outline of statistics*. McGraw Hill Professional, 2017.
24. Roger Stafford. Random vectors with fixed sum. Technical Report Available at https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum, MathWorks, 2006.

25. Rick S. H. Willemsen, Wilco van den Heuvel, and Michel van de Velden. Generating random vectors satisfying linear and nonlinear constraints, 2025. URL: https://arxiv.org/abs/2501.16936, arXiv:2501.16936.
26. Wufei Wu, Huijuan Huang, Wenhao Li, Ruihua Liu, Yong Xie, and Saiqin Long. Real-time analysis and message priority assignment for TSN-CAN gateway. *IEEE Trans. Intell. Transp. Syst.*, 25(11):16133–16144, 2024. doi:10.1109/TITS.2024.3425511.